Learning Network Event Sequences Using Long Short-term Memory and Second-order Statistic Loss

Hao Sha and **Mohammad Hasan** and **George Mohler**
Indiana University Purdue University Indianapolis
723 W Michigan St, SL 280
Indianapolis, Indiana 46202

Abstract

Modeling temporal event sequences on the vertices of a network is an important problem with widespread applications; examples include modeling influences in social networks, preventing crimes by modeling their space-time occurrences, and forecasting earthquakes. Existing solutions for this problem use a parametric approach, whose applicability is limited to event sequences following some well-known distributions, which is not true for many real life event datasets. To overcome this limitation, in this work, we propose a composite recurrent neural network model for learning events occurring in the vertices of a network over time. Our proposed model combines two long short-term memory units to capture base intensity and conditional intensity of an event sequence. We also introduce a second-order statistic loss that penalizes higher divergence between the generated and the target sequence's distribution of hop count distance of consecutive events. Given a sequence of vertices of a network in which an event has occurred, the proposed model predicts the vertex where the next event would most likely occur. Experimental results on synthetic and real-world datasets validate the superiority of our proposed model in comparison to various baseline methods.

1 Introduction

Modeling event sequences is essential in many areas, such as earthquake forecasting (Ogata 1998), crime prevention (Mohler et al. 2011; Short et al. 2014), and user-behavior study in social networks (Zhao et al. 2015; Mitchell and Cates 2010). Earthquakes tend to occur as sequences or clusters in close spatial and temporal proximity. Modeling earthquake sequences helps forecast future earthquakes and mitigate the seismic hazard. Space-time clustering event sequences are also observed in certain types of crime data, such as burglary and gang violence (Short et al. 2014). Studying these sequences may help identify crime patterns and prevent crimes from happening. Social networking services allow users to share content, and widely popular content can be shared by hundreds of thousands of users. Content sharing events form a sequence that spreads through social networks, as such, modeling this sequences could help predict content popularity, and provide useful information for content ranking and aggregation. It also helps better understanding of influence, fake-news or rumor propagation over the social networks.

An event sequence is a series of timestamp and mark pairs organized in time ascending order. The timestamps denote the time when events occur; while the marks indicate the identity of events. For instance, an event sequence could be a series of time and user ID pairs indicating when and who posts a photo. It may also be a sequence of time and longitude/latitude coordinates indicating when and where earthquakes occur. In many cases, sequential events occur at the vertices of an existing network or the events can be mapped to one of the vertices of a constructed network. For instance, earthquakes predominantly occur along fault lines, where tectonic movements are active. Therefore, earthquakes can be assigned membership to a large-scale fault-line cluster (Cheng, Dundar, and Mohler 2018). By inserting vertices on a fault line and then connecting nearby vertices, an earthquake network can be constructed. A sequence of earthquakes can thus be viewed as a series of events occurring on the nodes of this network. As another example, posting messages on an online social network (such as, Facebook or Twitter) can be viewed as an event occurring at a node of that network. In this work, we focus on modeling the event sequences in a network with an objective to predict the nodes where the future events are likely to occur and to generate sequences that closely resemble the real sequences.

The occurrence of an event may be spontaneous and independent of other events. On the other hand, it may also be triggered by the previous events (self-excitation). Traditionally, event sequences are modeled by various point processes (Odd Aalen and Gjessing 2008; Frank and Kingman 1993; Hawkes 1971). In particular, Hawkes processes (Hawkes 1971) model the spontaneity by a based intensity and the self-excitation by a time-varying conditional intensity. However, these models rely on some predefined parametric forms, thus limiting their capability of modeling arbitrarily distributed event data. To remedy this problem, EM-based nonparametric models (Lewis and Mohler 2011; Zhou, Zha, and Song 2013) are proposed. However, these models are still under the framework of Hawkes processes and may suffer from model mis-specification when the underlining event generation mechanism is not known a priori.

Alternatively, one could ignore the space-time coupling of event sequence and model an event's time and location in-

dependently. For instance, one can turn to Recurrent Neural Net (RNN) (Rumelhart, Hinton, and Williams 1988), specifically, Long Short-term Memory (LSTM) (Hochreiter and Schmidhuber 1997; Graves 2013) to model a general point process. Various models (Du et al. 2016; Xiao et al. 2017b; Mei and Eisner 2016) along this line are proposed, but in these models, the events' locations are not considered, and they are not necessarily occurring in a network node. One can also emphasize the network on which the events occur, ignoring their timestamp and consider event prediction as a node classification task, which can be solved by using a network embedding model (Zhang et al. 2018); however, such a direction ignores the temporal dependency of successive events, resulting in poor performance. Some existing works on network embedding consider temporal change in network (Zhu et al. 2016; Ma et al. 2018), but they are not appropriate for our task because in our task the underlying network is fixed, and we are merely interested in the temporal sequence of nodes where the future events will occur.

In this work, we propose a novel method for modeling temporal event sequence in the vertices of a network. Our proposed model uses LSTM to minimize the cross-entropy between the generated event probability and the one-hot encoding of the real event, which essentially enforces the generated sequences to have a first-order statistics similar to the real sequence. However, a potential next event may occur at any of the neighbors of the current event node and does not have to be the exact neighbor in the real sequence. The first-order statistic loss (cross-entropy loss) does not take into account this aspect. In contrast, the second-order loss penalizes the network distance between events rather than their identity, thus favoring the events that are within the correct hop distance in the network. For implementation, we combine two LSTMs—our first LSTM takes long-term event counts as inputs, while the second LSTM takes short-term event marks as inputs. For instance, we may feed the monthly event counts during the past 30 months to the first LSTM and the latest 30 events to the second LSTM. The first LSTM thus learns the slowly varying characteristics, while the second LSTM learns the fast-changing characteristics. Unlike existing works on point process (Xiao et al. 2017b), our model does not require domain-specific features and solely relies on the event sequences.

The contributions of this paper are listed below:

1. We propose combining two LSTMs to model both the slowly varying base intensity and the fast varying conditional intensity of an event sequence on a network.
2. We introduce a new loss function using the network distance distribution of consecutive events along with an illustration of its implementation, which is differentiable.
3. We compare our model with various baselines on both synthetic and real-world datasets to show the superiority of the proposed model for predicting the next node where the event will occur.

The rest of the paper is organized as follows: In Section 2, we provide some backgrounds regarding the important components of our model. In Section 3, we discuss related works in learning event sequence. In Section 4, we present a detailed description of our method. In Section 5, we describe the experiments and present the results. Finally, we summarize our work in Section 6.

2 Background

In this section, we briefly go through the major building components of this work and provide necessary background information.

2.1 Hawkes Process

Hawkes processes (Hawkes 1971) are self-exciting point processes where the occurrence of an event increases the likelihood of the occurrence of future events. Hawkes processes are characterized by an intensity function

$$\lambda_v(t) = \mu_v(t) + \sum_{\{(v_i,t_i)|t>t_i\}} g_v(v_i, t-t_i), \qquad (1)$$

where $\lambda_v(t)$ is the intensity of an event v at time t. $\mu_v(t)$ represents the base intensity. The triggering kernels $g_v(v_i, t-t_i)$ are accumulated over the historical events $\{v_i, t_i\}$. It is common to assume that the base intensity is constant over time $\mu_v(t) = \mu_v$ when the triggering kernels $g_v(v_i, t-t_i)$ vary significantly faster than $\mu_v(t)$. Furthermore, one may assume that the kernels have some predefined functional forms, such as the exponential function or the power-law function. With a constant base intensity and exponential kernels, Eq. 1 becomes

$$\lambda_v(t) = \mu_v + \sum_{\{(v_i,t_i)|t>t_i\}} W_{v,v_i} e^{-w(t-t_i)}, \qquad (2)$$

where $\mu_v \geq 0$ is the time-independent base intensity, w is the decay rate, and $W_{v,v_i} \geq 0$ is a measurement by which v_i initially excites v. Let $G = \{V, E\}$ denote a graph, where V and E are collections of vertices and edges. Let the subscript v in Eq. 2 denote an event occurring on node $v \in V$. If we assume that an event at node v is either spontaneous (determined by μ_v) or triggered homogeneously by its neighbors v_i, then we have $W_{v,v_i} = A_{v,v_i}$, where A is the adjacency matrix of G. We use the Hawkes process of Eq. 2 to generate synthetic sequences in experiments (Xu and Zha 2017). In addition, one of the baselines, multidimensional Hawkes process model (Xu and Zha 2017), is also based on this formula.

2.2 Long Short-term Memory Architecture

Recurrent Neural Net (RNN) (Rumelhart, Hinton, and Williams 1988) is a neural network model designed for modeling time series data. Besides taking input at each time step, it passes down the hidden state of the previous time step. Thus, it is capable of exhibiting temporal dynamic behaviors. However, as pointed out by (Bengio, Simard, and Frasconi 1994; Pascanu, Mikolov, and Bengio 2013), traditional RNNs suffer from vanishing (exploding) gradient problem, preventing them from learning relationships separated by an extended period. To remedy this problem, authors of (Hochreiter and Schmidhuber 1997) propose a Long

Short-term Memory (LSTM) architecture where a cell state is introduced. Now the output and the intermediate states at time t are collectively determined by the input, the hidden state and the cell state at time $t-1$. Such design effectively reduces the multiplicative effect of the small gradients. In this work, we adopt the version of LSTM implemented by the following equations:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{V}_i\mathbf{c}_{t-1} + \mathbf{b}_i),\\
\mathbf{f}_t &= \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{V}_f\mathbf{c}_{t-1} + \mathbf{b}_f),\\
\mathbf{c}_t &= \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t \odot tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{b}_c),\\
\mathbf{o}_t &= \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{V}_o\mathbf{c}_t + \mathbf{b}_o),\\
\mathbf{h}_t &= \mathbf{o}_t \odot tanh(\mathbf{c}_t),
\end{aligned} \tag{3}$$

where σ is the sigmoid function and $\odot$ is the Hadamard product. $\mathbf{i}_t$, $\mathbf{f}_t$, and $\mathbf{o}_t$ are vectors of intermediate states; while $\mathbf{x}_t$, $\mathbf{h}_t$, and $\mathbf{c}_t$ are vectors of input, hidden, and cell states, respectively. Matrices $\mathbf{W}_*$, $\mathbf{U}_*$, $\mathbf{V}_*$, and vectors $\mathbf{b}_*$ are trainable parameters.

In recent years, LSTM has become a popular choice for modeling sequential data, along with other methods such as dilated causal convolutions (van den Oord et al. 2016) and Gated Recurrent Unit (GRU) (Cho et al. 2014). LSTM has been successfully applied to solve various sequence related learning problems, such as, speech recognition (Graves and Jaitly 2014), language translation (Sutskever, Vinyals, and Le 2014), handwriting synthesis (Graves 2013), and image generation (Gregor et al. 2015).

2.3 Second-order Statistics of Sequential Events

Besides correlating in time, sequential events may also correlate in types or space. For instance, aftershocks commonly occur in the vicinity of a major earthquake. A picture posted by a user is more likely to be liked by a friend than a stranger. The general spatial correlation could be measured by Ripley's K-function (Ripley 1976; Dixon 2014) which is defined as:

$$\begin{aligned}
K(d) =& \lambda^{-1}E[number\ of\ extra\ events\ within\\
& distance\ d\ of\ a\ randomly\ chosen\ event],
\end{aligned} \tag{4}$$

where λ is the event density (number per unit area). If edge effects are ignored, $K(d)$ can be estimated by (Dixon 2014):

$$\hat{K}(d) = \frac{\sum_{i,j}\mathbb{1}(d_{i,j} < d)}{N\lambda}, \tag{5}$$

where N is the number of events and $d_{i,j}$ is the distance between the i'th and j'th events. $\mathbb{1}(x)$ is the indicator function with the value 1 if x is true and 0 otherwise. For network events, $d_{i,j}$ can be defined as the length of the shortest path between node v_i and v_j where the i'th and j'th events occur. Based on this concept, we propose using a normalized K_1 estimate to describe the second-order property of a sequence in a network. In specific, given a sequence $S = \{t_i, v_i\}$ in a network $G = \{V, E\}$ with $v_i \in V$, we define the following K_1 estimate:

$$K_1(d_m|S,G) = \frac{\sum_{i=1}^{N-1}\mathbb{1}(d_{i,i+1} < d_m)}{\sum_{d_m=0}^{d}\sum_{i=1}^{N-1}\mathbb{1}(d_{i,i+1} < d_m)} \tag{6}$$

where N is the size of the sequence and d is the diameter of the network. d_m can take discrete values from 0 to d, representing the range of all possible network distances between any two nodes. $K_1(d_m|S,G)$ is essentially an estimate of the distribution of the distance between two consecutive events in the network G. Assuming $S' = \{t'_i, v'_i\}$ is an artificial sequence generated by some model in the same network G, we can enforce S' to have the same distance distribution as S by minimizing $D(K_1(d_m|S,G)||K_1(d_m|S',G))$, where $D(P||Q)$ measures the distance between distribution P and Q (such as Kullback-Leibler divergence, Jensen-Shannon divergence and L2-norm).

3 Related Works

Event sequences have been primarily modeled by point processes (Odd Aalen and Gjessing 2008), where predefined intensity functions are used to capture the generative mechanisms. For instance, Hawkes processes (Hawkes 1971) model the self/mutual-excitation effect by a time-varying conditional intensity, which is boosted upon the arrival of a new event (Eq. 1). Point processes have been widely applied to the areas such as seismology (Ogata 1998; Marsan and Lengliné 2008), criminology (Mohler et al. 2011; Short et al. 2014), social activity analysis (Farajtabar et al. 2014; Zhao et al. 2015), and information diffusion (Du et al. 2013a; 2013b). Despite their success, parametric point process models (such as Eq. 2) have some limitations. Most notably, their intensities rely on explicitly defined kernel functions. An inappropriate selection of the kernel function may cause significant degradation of the model. To remedy the problem, (Lewis and Mohler 2011) proposes an EM-based nonparametric model to estimate the intensity functions without prior knowledge of their form. Furthermore, (Zhou, Zha, and Song 2013) and (Luo et al. 2015) extend this method to the multidimensional Hawkes processes. Despite their enhanced flexibility, these models still rely on the assumption that the events are generated by a Hawkes process, and may not perform well for other point processes.

For better generalization, (Du et al. 2016; Xiao et al. 2017b) resort to RNN to model arbitrarily distributed event data. In particular, (Xiao et al. 2017b) combines two LSTMs that take synchronized time series as well as asynchronous event sequences as inputs to predict both timestamps and marks. But, in their model, the event process does not occur in the vertices of a network. Besides, their model relies on domain-specific features such as ATM logs, which are often not available. Alternatively, (Xiao et al. 2017a) replaces RNN with Generative Adversary Net (GAN) (Goodfellow et al. 2014; Goodfellow, Shlens, and Szegedy 2014), specifically, Wasserstein GAN (Arjovsky, Chintala, and Bottou 2017). However, their model is specialized in unmarked temporal point processes and is not capable of predicting event types. Another line of researches (Xu, Farajtabar, and Zha 2016) focuses on inferring the causal network among

Key variables in this work. $|V|$ is the number of vertices in the graph.

symbol	size	description
k_l	1	number of cells in an LSTM1 layer
k_s	1	number of cells in an LSTM2 layer
d_h	1	LSTM hidden dimension
$\mathbf{y}$	$\|V\| \times 1$	one-hot encoding of a real event
$\mathbf{x}^l$	$k_l \times 1$	event count vector
$\mathbf{x}^s$	$k_s \times 1$	embedding vector of an event
$\mathbf{u}$	$\|V\| \times 1$	predictive distribution
$\mathbf{h}$	$d_h \times 1$	LSTM hidden state
$\mathbf{c}$	$d_h \times 1$	LSTM cell state
$\mathbf{D}$	$\|V\| \times \|V\|$	network distance matrix
$\mathbf{W}$	$\|V\| \times (\|V\| + d_h)$	weight (fully-connected layer)
$\mathbf{b}$	$\|V\| \times 1$	bias (fully-connected layer)

different types of events. Our model is different from these models as our networks are constructed beforehand; besides, our causal networks are significantly larger.

4 Methods

4.1 Problem Description

Given a network $G = \{V, E\}$ where V and E are the collections of vertices and edges, we can represent a sequence of N events on the vertices V before time T by a series of vertex-time pairs, $S = \{(v_i, t_i) | v_i \in V, t_i \in [0, T), t_i < t_{i+1}, i = 1, ..., N\}$. Furthermore, we can extract the vertices from S to form a vertex sequence $\hat{S} = \{v_i | v_i \in V, i = 1, ..., N\}$. In this work, we focus on learning a model to predict the vertex where the next event is most likely to occur based on the previous events. Formally, given a graph $G = \{V, E\}$ and a sequence $\hat{S} = \{v_i | v_i \in V, i = 1, ..., N\}$, we denote

$$u^v_{N+1} = P(v|\hat{S}) \tag{7}$$

where u^v_{N+1} is the probability for the $(N+1)$'th event to occur on vertex $v \in V$, and $P(v|\hat{S})$ is the model. We can sample the next event via $v \sim P(v|\hat{S}), v \in V$, and we call this task **event prediction**. Assuming that v'_{N+1} is the $(N+1)$'s event predicted by the model, we can append it to the end of $\hat{S}$, to obtain an extended sequence $\hat{S}' = [\hat{S}, v'_{N+1}]$. We then feed $\hat{S}'$ to Eq. 7 to predict the $(N+2)$'th event. By repeating this process, we can generate an artificial sequence of arbitrary length on the graph, and we call this task **sequence generation**. Some key variables adopted in the following section are listed in Table 4.1.

4.2 Model Formulation

We propose an architecture consisting of two LSTMs, denoted by LSTM1 and LSTM2. Fig. 1 is an illustration of this architecture. Both LSTMs (shaded areas in Fig. 1) are deep (multi-layer) and contain a stack of recurrently connected memory cells, each of which performs the calculations in Eq. 3. Note that the number of layers and cells in either LSTM are adjustable and not necessarily the same as those in Fig. 1. Summarized in Eq. 8, a cell in LSTM1 (LSTM2) takes an input $\mathbf{x}^l_i$ ($\mathbf{x}^s_i$), a hidden state $\mathbf{h}^l_i$ ($\mathbf{h}^s_i$) and a cell state $\mathbf{c}^l_i$ ($\mathbf{c}^s_i$), and outputs a new hidden state $\mathbf{h}^l_{i+1}$ ($\mathbf{h}^s_{i+1}$) and a new cell state $\mathbf{c}^l_{i+1}$ ($\mathbf{c}^s_{i+1}$). As shown in Fig. 1, these

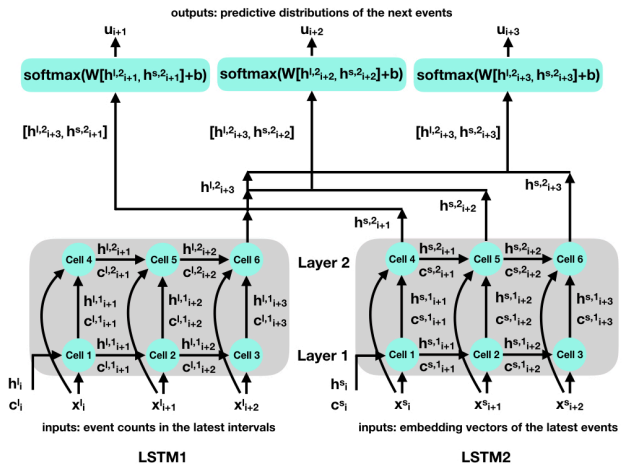


Figure 1: Network Architecture. The shaded areas represent LSTMs; while the cyan circles represent cells. Each LSTM contains two layers.

new states are then passed horizontally to the next cell in the same layer and vertically to the cell in the next layer. The final outputs of the two LSTMs are concatenated and fed to a fully-connected layer with a Softmax activation, to generate a predictive distribution $\mathbf{u}_{i+1}$ over the vertices (V) for the next event (Eq. 8).

$$\begin{aligned}
(\mathbf{h}^l_{i+1}, \mathbf{c}^l_{i+1}) &= cell_{LSTM1}(\mathbf{x}^l_i, \mathbf{h}^l_i, \mathbf{c}^l_i) \\
(\mathbf{h}^s_{i+1}, \mathbf{c}^s_{i+1}) &= cell_{LSTM2}(\mathbf{x}^s_i, \mathbf{h}^s_i, \mathbf{c}^s_i) \\
\mathbf{e}_i &= [\mathbf{h}^l_f, \mathbf{h}^s_{f,i}] \\
\mathbf{u}_{i+1} &= Softmax(\mathbf{W}\mathbf{e}_i + \mathbf{b})
\end{aligned} \tag{8}$$

LSTM1 takes long-term event counts as input and learns the slowly varying background rate (similar to $\mu_v(t)$ in Eq. 1). We define an event count vector $\mathbf{x}^l_i = [x^l_{i,v}], v \in V$, with $x^l_{i,v}$ representing the number of events on node v during $[(i-1)\Delta t, i\Delta t)$. The inputs of LSTM1 are $\mathbf{x}^l_i$, $\mathbf{x}^l_{i+1}$, $\mathbf{x}^l_{i+2}$, ..., $\mathbf{x}^l_{i+k_l-1}$, corresponding to the latest k_l intervals. Note these pre-computed vectors are not trainable. LSTM2, on the other hand, learns the short-range correlation (resembling the triggering kernels in Eq. 1). Given that the latest k_s events occur on nodes v_i, v_{i+1}, ..., v_{i+k_s-1}, the inputs of LSTM2 are their embedding vectors $\mathbf{x}^s_i$, $\mathbf{x}^s_{i+1}$, $\mathbf{x}^s_{i+2}$, ..., $\mathbf{x}^s_{i+k_s-1}$. Note these embedding vectors are learned during training by back-propagation.

Let $\mathbf{h}^l_f$ denote the hidden state of the last cell in the last layer of LSTM1, and $\{\mathbf{h}^s_{f,i}, \mathbf{h}^s_{f,i+1}, ..., \mathbf{h}^s_{f,i+k_s-1}\}$ denote the hidden states in the last layer of LSTM2. Summarized in the last two equations in Eq. 8, concatenating $\mathbf{h}^l_f$ to each element in $\{\mathbf{h}^s_{f,i}, \mathbf{h}^s_{f,i+1}, ..., \mathbf{h}^s_{f,i+k_s-1}\}$, we obtain a series of vectors $\{\mathbf{e}_i, \mathbf{e}_{i+1}, .., \mathbf{e}_{i+k_s-1}\}$, which are then fed to the Softmax layer to generate a series of vectors $\{\mathbf{u}_{i+1}, \mathbf{u}_{i+2}, ..., \mathbf{u}_{i+k_s}\}$, representing the probability distributions over the vertices for the next events. Let $\{\mathbf{y}_{i+1}, \mathbf{y}_{i+2}, ..., \mathbf{y}_{i+k_s}\}$ denote the one-hot encoding of the nodes where the real events occur. The cross-entropy loss can thus be written as the following:

$$L_C = -\frac{1}{k_s}\sum_{k=1}^{k_s}\sum_{j=1}^{|V|} y_{i+k}^j log(u_{i+k}^j), \quad (9)$$

where y_{i+k}^j and u_{i+k}^j are the j'th elements of $\mathbf{y}_{i+k}$ and $\mathbf{u}_{i+k}$, respectively. The summation is over the number of steps k_s and the number of nodes $|V|$ in the network.

As mentioned in Sec. 2.3, by minimizing $D(K_1(d_m|S,G)||K_1(d_m|S',G))$, we could ensure that the generated sequence S' has the same distance distribution as the real sequence S. However, implementing K_1 in a neural network setting is not trivial. In particular, the right-hand side of Eq. 6 is not differentiable. If our loss function contains Eq. 6, we would not be able to perform gradient descent. To overcome this problem, we smooth Eq. 6 using Kernel Density Estimation (KDE) with a Gaussian kernel:

$$K_1(d_m|S,G) = \frac{\sum_{i=1}^{N-1} exp(-(\frac{d_{i,i+1}-d_m}{h})^2)}{\sum_{d_m=0}^{d}\sum_{i=1}^{N-1} exp(-(\frac{d_{i,i+1}-d_m}{h})^2)}, \quad (10)$$

where $h > 0$ is the bandwidth. With a small enough h, Eq. 10 is a good approximation to Eq. 6. Most importantly, Eq. 10 is differentiable and thus can be used to construct a loss function.

Furthermore, we can calculate $d_{i,i+1}$ efficiently by matrix multiplication. We precompute a distance matrix $\mathbf{D}$ with entry $D_{i,j}$ representing the distance between node v_i and v_j. Let $d_{i,i+1}$ denote the distance between the real events v_i and v_{i+1} and $d'_{i,i+1}$ denote the distance between the generated events v'_i and v'_{i+1}, then we have

$$\begin{aligned} d_{i,i+1} &= \mathbf{y}_i^T \mathbf{D} \mathbf{y}_{i+1}, \\ d'_{i,i+1} &= \mathbf{u}_i^T \mathbf{D} \mathbf{u}_{i+1}, \end{aligned} \quad (11)$$

where $\mathbf{y}_i$ and $\mathbf{y}_{i+1}$ are the one-hot encoding of the real events at step i and $i+1$, and $\mathbf{u}_i$ and $\mathbf{u}_{i+1}$ are predictive distributions over events at step i and $i+1$.

Plugging Eq. 11 into Eq. 10, we obtain

$$K_1(d_m|S,G) = \frac{\sum_{i=1}^{N-1} exp(-(\frac{\mathbf{y}_i^T \mathbf{D} \mathbf{y}_{i+1}-d_m}{h})^2)}{\sum_{d_m=0}^{d}\sum_{i=1}^{N-1} exp(-(\frac{\mathbf{y}_i^T \mathbf{D} \mathbf{y}_{i+1}-d_m}{h})^2)}, \quad (12)$$

for a real sequence S, and

$$K_1(d_m|S',G) = \frac{\sum_{i=1}^{N-1} exp(-(\frac{\mathbf{u}_i^T \mathbf{D} \mathbf{u}_{i+1}-d_m}{h})^2)}{\sum_{d_m=0}^{d}\sum_{i=1}^{N-1} exp(-(\frac{\mathbf{u}_i^T \mathbf{D} \mathbf{u}_{i+1}-d_m}{h})^2)}, \quad (13)$$

for a generated sequence S'.

Accordingly, we define the following loss function:

$$L_K(S,S'|G) = D(K_1(d_m|S',G)||K_1(d_m|S,G)), \quad (14)$$

where $D(P||Q)$ could be KL-divergence, JS-divergence or L2-norm. Overall, we propose a total loss function of the following form:

$$\begin{aligned} L &= L_C + \lambda L_K \\ &= -\frac{1}{k_s}\sum_{k=1}^{k_s}\sum_{j=1}^{|V|} y_{i+k}^j log(u_{i+k}^j) \\ &+ \lambda D(K_1(d_m|S',G)||K_1(d_m|S,G)), \end{aligned} \quad (15)$$

where λ is a hyper-parameter for regularization. k_s is the number of cells in a layer of LSTM2. The second term on the right (regularization) is obtained by plugging Eq. 12 and 13 into Eq. 14.

Composite LSTM with Second-order Statistic Loss

Require: $\mathbf{x}_i^l$, $\mathbf{x}_i^s$, $\mathbf{y}_i$, $\mathbf{h}_i$, $\mathbf{c}_i$, $\mathbf{D}$, λ, number of epochs n, epoch size m, batch size o

Ensure: model M with optimal set of parameters Θ

for epoch = 1 to n **do**
  for batch = 1 to m **do**
    for i = 1 to o **do**
      Initialize $\mathbf{h}_0^l = \mathbf{c}_0^l = \mathbf{h}_0^s = \mathbf{c}_0^s = \mathbf{0}$
      $\mathbf{H}_i^l \leftarrow LSTM1(\mathbf{x}_i^l, ..., \mathbf{x}_{i+k_l-1}^l, \mathbf{h}_0^l, \mathbf{c}_0^l)$
      $\mathbf{h}_{i+1}^l, ..., \mathbf{h}_{i+k_l}^l \leftarrow \mathbf{H}_i^l$
      $\mathbf{H}_i^s \leftarrow LSTM2(\mathbf{x}_i^s, ..., \mathbf{x}_{i+k_s-1}^s, \mathbf{h}_0^s, \mathbf{c}_0^s)$
      $\mathbf{h}_{i+1}^s, ..., \mathbf{h}_{i+k_s}^s \leftarrow \mathbf{H}_i^s$
      for j = 1 to k_s **do**
        $\mathbf{e}_{i+j} \leftarrow Concatenate(\mathbf{h}_{i+k_l}^l, \mathbf{h}_{i+j}^s)$
        $\mathbf{u}_{i+j} \leftarrow Softmax(\mathbf{W}\mathbf{e}_{i+j} + \mathbf{b})$
        $d_{i+j-1,i+j} \leftarrow \mathbf{y}_{i+j-1}^T \mathbf{D} \mathbf{y}_{i+j}$
        $d'_{i+j-1,i+j} \leftarrow \mathbf{u}_{i+j-1}^T \mathbf{D} \mathbf{u}_{i+j}$
      end for
    end for
    Calculate loss L using Eq. 15 and update $\mathbf{W}$, $\mathbf{b}$, embedding, and the trainable parameters in Eq. 3 using gradient descent.
  end for
end for

4.3 Training Protocol

In Algorithm 4.2, we show the pseudo-code of our training procedure. A sample of input contains k_s embedding vectors $\mathbf{x}_i^s$ representing the current window of events. It also contains k_l event count vectors $\mathbf{x}_i^l$ that provide the background information of the current window. At the beginning of each window, the hidden states and the cell states are initialized as zero vectors, which are passed to the LSTMs along with the inputs at various steps (intervals). The final hidden state ($\mathbf{h}_{i+k_l}^l$) of LSTM1 is concatenated with each of the k_s hidden state of LSTM2. The resulting vectors $\mathbf{e}_{i+j}$ then go through a fully-connected layer with Sofmax activation, to generate the predictive distributions $\mathbf{u}_{i+j}$. Meanwhile, the network distances between consecutive events, $d_{i+j-1,i+j}$ (real) and $d'_{i+j-1,i+j}$ (predicted) are retrieved from a precomputed hop-distance matrix $\mathbf{D}$. At the end of a batch, the total loss function is computed using Eq. 15 and its gradient is evaluated and averaged over the entire batch. The gradient descent optimizer is adopted to update the trainable parameters, including the embedding, the parameters in the LSTMs and those in the fully-connected layer.

5 Experiment

We run several experiments to validate the effectiveness of our proposed model and to compare the performance of our

Table 1: Dataset properties. $|V|$ and $|E|$ are the number of nodes and number of edges of each network, respectively. Sequence size gives the number of events in each sequence.

dataset	$\|V\|$	$\|E\|$	sequence size
Rand-1	291	290	10000
Rand-2	1599	1940	10000
Earthquake	648	41744	17381
Email	2634	6458	14092
Twitter	393	777	18879

model over a set of competing models. In our first experiment, we perform **event prediction**, using the proposed model to predict the vertices of a network in which future events will occur. In detail, we feed event counts at each vertex in the past 30 ($k_l = 30$) successive intervals $\{\mathbf{x}_i^l, \mathbf{x}_{i+1}^l, ..., \mathbf{x}_{i+29}^l\}$ to LSTM1 and the embedding vectors of 32 ($k_s = 32$) historical events $\{\mathbf{x}_i^s, \mathbf{x}_{i+1}^s, ..., \mathbf{x}_{i+31}^s\}$ to LSTM2. Note that the selections of k_l and k_s are heuristic. In general, using values too small might reduce the model's capacity, whereas using values too large might make the training very expensive. The output $\mathbf{u}_{i+32}$ gives a probability distribution for an event to occur at each vertex at step $i + 32$. To evaluate our predictions, we calculate a series of hit rates (hit@10, hit@20, and hit@30). For instance, if the true event v_{i+32} is among the top 10 most probable vertices given by $\mathbf{u}_{i+32}$, we consider it as a hit. We slide a window of 32 events through the entire test set and obtain the average hit rates to indicate the prediction performance.

Our second experiment is **sequence generation**. Similar to what we did in event prediction, 30 ($k_l = 30$) event count vectors and 32 ($k_s = 32$) embedding vectors are fed to LSTM1 and LSTM2, to generate a predictive distribution for the next event. However, here we sample the next event at step $i+32$ from a multinomial distribution given by $\mathbf{u}_{i+32}$. Assuming that the event that we draw is v'_{i+32}, we then append it to the past 31 events, resulting a new list of 32 events $\{v_{i+1}, v_{i+2}, ..., v'_{i+32}\}$. Here we essentially keep a queue of 32 successive events, where the oldest event leaves the queue when a newly sampled event enters the queue. We then feed the embedding vectors and event count vectors of the current queue to the model to obtain the probability distribution for the next event at step $i + 33$. By repeating this process, we can generate a fake sequence of arbitrary length. To see how realistic the generated sequences are, we compare them with the real sequences in terms of diffusion pattern.

We also perform additional experiments for validating model convergence, and robustness, which are presented in the Supplement. Below, we first discuss the datasets on which we run our experiments and the competing models with which we compare our model.

5.1 Data Description

We use three real-world datasets, Earthquake, Email and Twitter, and two synthetic datasets, Rand-1, and Rand-2. Each of these is a composition of a graph and a node sequence on which an event occurred. For all datasets, 70% of the sequence from its prefix is used for training and the remaining part of the sequence is used for test. Statistics of the dataset is provided in Table 1. More discussion of the datasets is provided below:

Earthquake: (SCEDC 2013) contains the location, time and magnitude of earthquakes that occurred in Southern California. We construct a network based on the Community Fault Model 3.0 (Plesch and others 2007), which is a 3D representation (latitude, longitude, and elevation) of faults in Southern California. Specifically, we sample every 100 points from each fault line and add an edge between two points if their distance is less than 40 kilometers. The resulting network contains 648 nodes and 41744 edges (Table 1). We collect earthquakes from 1997 to 2018 with a magnitude of at least 2.5 and map them to the nearest location (node) in the network. Consequently, we obtain a sequence of 17381 earthquakes that occur on the nodes of the fault network.

Email: Enron email is a publicly available dataset that contains $\sim 500,000$ emails generated by employees of the Enron Corporation. We use email addresses owned by Enron employees as nodes, and add an edge between two nodes if at least one email has been exchanged between the corresponding addresses. Since the resulting network is not connected, we select the largest connected component that contains 2634 nodes and 6458 edges as our final network (Table 1). We extract the sender address and timestamp from each email across the entire corpse. The sender-time pairs are then sorted in time ascending order. The resulting sequence contains 14092 email sending events (sender-time pairs).

Twitter: This dataset contains Twitter data collected during the presidential election in South Africa in 2014. We define a tweet as popular if it is retweeted more than 10 times, and a user as popular if she has posted a popular tweet. We use the popular users as nodes to construct a network. We add an edge (undirected) between two nodes if one of them has mentioned the other in a popular tweet. The resulting network is not connected. We thus select the largest connected component that has 393 nodes and 777 edges (Table 1). We consider that an event occurs on a node when the corresponding user posts a tweet. We gather a series of such events in time ascending order. The resulting sequence contains 18879 events (user-time pairs).

Rand-1 & Rand-2: We use Erdős-Rényi model to generate random graphs $G(n, p)$ with n nodes, where an edge is connected randomly with probability p independent of every other edge. First, we generate two random graphs $G(n = 2000, p = 0.0001)$ and $G(n = 2000, p = 0.001)$, then we extract the largest connected components from these random graphs, rendering two connected graphs with ($|V| = 291$, $|E| = 290$) and ($|V| = 1599$, $|E| = 1940$), respectively (Table 1). Next, we simulate two event sequences on these two graphs using multi-dimensional Hawkes Process (Xu and Zha 2017). Specifically, these sequences are generated with predefined base intensities $\mu_0^v = 10^{-3}u/|V|, u \sim U(0, 1), v \in V$ and decay rate $w = 1$. We use adjacency matrix for W_{v,v_i} in Eq. 2. Both sequences contain $N = 10^4$ events.

5.2 Competing Methods

We compare our proposed model with the following competing methods. All methods were given the identical experimental setup (same data input) to maintain fairness.

Table 2: Experimental Results.

	Rand-1			Rand-2			Earthquake			Email			Twitter		
Model	Hit@10	Hit@20	Hit@30	Hit@10	Hit@20	Hit@30	Hit@10	Hit@20	Hit@30	Hit@10	Hit@20	Hit@30	Hit@10	Hit@20	Hit@30
LC	0.527	0.768	0.879	0.237	0.352	0.437	0.638	0.715	0.756	0.293	**0.412**	0.465	0.652	0.708	0.740
LC + LK	**0.532**	**0.769**	**0.886**	**0.241**	**0.358**	**0.442**	**0.648**	**0.722**	**0.768**	**0.306**	0.411	**0.467**	**0.655**	**0.711**	**0.742**
Node2Vec-dense	0.420	0.655	0.786	0.148	0.251	0.337	0.494	0.589	0.672	0.176	0.248	0.298	0.533	0.603	0.650
Node2Vec-cnn	0.449	0.724	0.872	0.165	0.307	0.410	0.452	0.585	0.649	0.162	0.233	0.282	0.507	0.555	0.605
DeepWalk-dense	0.403	0.627	0.756	0.150	0.245	0.331	0.526	0.625	0.677	0.182	0.256	0.309	0.540	0.610	0.655
DeepWalk-cnn	0.423	0.697	0.863	0.168	0.292	0.384	0.478	0.579	0.667	0.148	0.233	0.293	0.511	0.580	0.631
Random-Walk	0.013	0.016	0.066	0.005	0.015	0.016	0.300	0.364	0.370	0.003	0.009	0.013	0.143	0.159	0.175
Hawkes-Exp	0.503	0.721	0.827	0.196	0.302	0.362	0.622	0.672	0.697	0.257	0.325	0.355	0.599	0.640	0.664
RNNPP	0.348	0.717	0.826	0.208	0.308	0.375	0.376	0.493	0.554	0.249	0.359	0.406	0.587	0.649	0.692
Logistic	0.005	0.113	0.147	0.015	0.029	0.037	0.122	0.197	0.200	0.002	0.004	0.005	0.008	0.025	0.045

DeepWalk: DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) learns latent representations of nodes in a network using truncated random walks. We apply DeepWalk to each of the networks with the embedding dimension in $\{64, 128, 256\}$, the walk length in $\{20, 40, 60\}$, and the window size in $\{5, 10, 15\}$. We adopt Multilayer Perceptron (MLP) (hidden dimension $\{256, 128\}$) with Softmax activation as the classifier in our **DeepWalk-dense** model. The classifier takes the embedding vectors of 32 historical events, and outputs the probability distribution of the next event. Additionally, we use a Convolutional Neural Network (CNN) as the classifier in our **DeepWalk-cnn** model. The classifier contains two CNN layers (hidden dimension $\{256, 128\}$), and two fully-connected layers (hidden dimension $\{100, |V|\}$). We use a filter size of 3, a stride of 1, and a max-pooling size of 2. We apply dropout of 0.5 in both DeepWalk-dense and DeepWalk-cnn.

Node2Vec: Node2Vec (Grover and Leskovec 2016) uses biased random walks to learn node embeddings in a network. We apply Node2Vec to each of the networks with the embedding dimension in $\{64, 128, 256\}$, and the return parameter p and the in-out parameter q in $\{0.5, 1.0., 1.5\}$. Similar to DeepWalk, we build classifiers using MLP (**Node2Vec-dense**) and CNN (**Node2Vec-cnn**). The classifiers have the same architecture and hyper-parameters as those of DeepWalk.

Random Walk: Given the current event, the next event is predicted by performing a random walk. Precisely, let v_i denotes the current node, the next node v_{i+1} is chosen uniformly at random from $N(v_i) \bigcup \{v_i\}$, where $N(v_i)$ is the set of neighbors of v_i.

Hawkes Processes (Hawkes-Exp): We fit a multi-dimensional Hawkes process using Eq. 2 with constant base intensity μ_0 and exponential kernels. In specific, we use a maximum likelihood estimator (MLE) with a sparse-group-lasso regularizer (Xu and Zha 2017). We test all combinations of hyper-parameters given decay rate w in $\{0.1, 0.5, 1.0\}$ and regularizer in $\{0.1, 1.0, 10.0\}$. For better performance, the original sequences are cut into subsequences based on constant time intervals. For a fair comparison, we feed the Hawkes process model with 32 historical events, the same as the number of cells in LSTM2. Therefore, the results of Hawkes-Exp should not be considered as the upper limit for Rand-1 or Rand-2, despite that they were generated using Hawkes processes.

RNN for Point Processes (RNNPP): (Xiao et al. 2017b) is an RNN model consist of two LSTMs that combine synchronized time series and asynchronous event sequences. The model is primarily designed for maintenance support services problem and relies on domain-specific features such as ATM logs. Without domain-specific features, we use sequences of constant features (e.g. vectors of ones) instead.

Logistic Classification (Logistic): The model is a multi-class logistic regression classifier with cross-entropy loss. It virtually fits $Softmax(\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{y}$ and resembles the last layer of our LSTM model (Fig. 1); $\mathbf{x}$ contains event counts on each node in a given window, and $\mathbf{y}$ indicates the identity of the next event.

5.3 Hyper-parameter Tuning

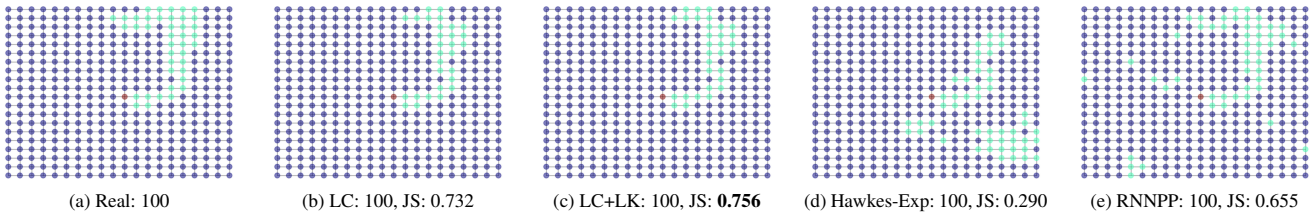
In the experiments, we tune the following hyper-parameters across all the datasets: λ in $\{0.1, 1, 10, 100, 1000\}$; learning-rate in $\{0.01, 0.1, 1.0, 10\}$ with a decay rate of 0.9; embedding dimension in $\{64, 128, 256, 512\}$. For the LC+LK model, the results shown in Table 2 are obtained using the optimal parameters (see Supplement for details). For the LC model, the hit rates are obtained using the same parameters as the LC+LK model without the second-order constraint.

The rest of the parameters are fixed. We use a batch-size of 32 and a dropout of 0.5. The number of layers in each LSTM is 2 and the numbers of cells in a layer of LSTM1 and LSTM2 are 30 and 32, respectively. Event counts are calculated using time intervals of 0.01 seconds (Rand-1 and Rand-2), 1 day (Email and Twitter), and 30 days (Earthquake). We adopt KL-divergence for all the datasets to evaluate the distance between two distributions (Eq. 14).

5.4 Results

Event prediction. In Table 2, we show the comparison results of the event prediction task for ten methods over five datasets using hit rate (@10, @20, and @30) as the evaluation metric. In the table, LC is our basic LSTM model, while LC + LK is our LSTM model with the second-order statistic loss. Remaining eight are competing methods. As we can see from the table, for all the datasets and for all different hit rates, LC + LK is the best method (except for Email for hit@20). Our proposed LC and LC + LK models win over Hawkes-Exp even for the random datasets in which events actually follow a Hawkes distribution. We can also see that our models outperform the embedding based methods where latent representations of the nodes are learned from the network topology. Our LC+LK model's performance is better than that of the LC model in all cases (except for Email for hit@20), which suggests that adding the second-order statistic loss can improve the model for predicting the future event.

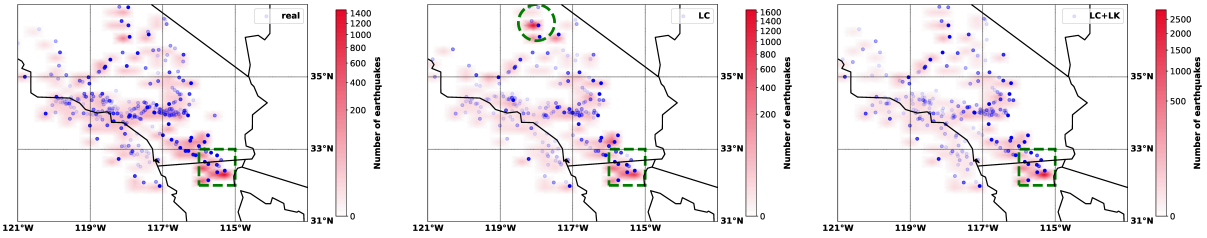
Sequence generation. We also test if our model can learn the correct diffusion of the real sequence. We simulate a se-



(a) Real: 100 (b) LC: 100, JS: 0.732 (c) LC+LK: 100, JS: **0.756** (d) Hawkes-Exp: 100, JS: 0.290 (e) RNNPP: 100, JS: 0.655

(f) Real: 1000 (g) LC: 1000, JS: 0.732 (h) LC+LK: 1000, JS: **0.753** (i) Hawkes-Exp: 1000, JS: 0.371 (j) RNNPP: 1000, JS: 0.428

Figure 2: Diffusion on grids. The sequences start from the center (red) of the grids. The green dots represent the nodes where events have occurred; while the purple dots represent the grid. The top and bottom rows are snapshots taken at time step 100 and 1000, respectively. Images from left to right represent different models. JS denotes the Jaccard Similarity between a generated sequence and a real sequence for each snapshot.



(a) Real (b) LC, corr: 0.515 (c) LC+LK, corr: **0.936**

Figure 3: Earthquakes in Southern California. Blue circles represent earthquake locations; red heatmaps indicate the number of earthquakes at each location. The correlation is between the event count distributions of the real and generated sequences.

Jaccard Similarity between a generated sequence and a real sequence for snapshots at different steps. The scores are in the form of mean +/- standard deviation, which are estimated over 100 experiments.

Step	LC	LC+LK	Hawkes-Exp	RNNPP
100	**0.754+/-0.108**	0.747+/-0.098	0.298+/-0.030	0.656+/-0.019
500	0.661+/-0.061	**0.673+/-0.058**	0.282+/-0.058	0.333+/-0.018
1000	0.755+/-0.039	**0.756+/-0.045**	0.435+/-0.066	0.414+/-0.018

quence on a 20×20 grid such that the event marks (nodes) are determined by a simple symmetric random walk and the event timestamps are determined by a homogeneous Poisson process with a rate of 10. We train various models with this sequence and generate fake sequences using them. We illustrate the diffusion processes in Fig. 2. The snapshots are taken at the 100'th and 1000'th time-step. The green nodes represent the nodes that have been visited by the sequences, while the purple nodes represent the grid. We calculate the Jaccard Similarity (JS under each image) between a generated sequence and a real sequence for each snapshot. A higher score indicates that the generated sequence better mimics the real sequence. In the snapshots, LC (2nd column) and LC+LK (3rd column) show similar patterns as the real sequence (1st column). In contrast, the last two columns suggest that Hawkes-Exp may be too conservative; while RNNPP may be too aggressive. In Table 5.4, we list the mean and standard deviation of Jaccard Similarity scores over 100 experiments. The snapshots are sampled at step 100, 500, and 1000. We can see that our LC and LC+LK models outperform the other two methods significantly. Notably, the LC+LK model scores the highest mean Jaccard Similarity in two out of the three cases. Overall, our models generate more realistic sequences than the competing methods, and by adding the second-order statistic constraint, the LC+LK model better captures the characteristics of the real sequence than the LC model.

Case study. We show the effectiveness of our model in the earthquake forecasting task. As mentioned in the previous section, we map earthquakes in Southern California during $1997 \sim 2018$ to a network constructed based on the fault lines (CFM3). After training, we use both LC and LC+LK models to generate fake sequences of the same length as the test data. In Fig. 3, we mark the locations of earthquakes (blue circles) on the map of Southern California. From left to right, the figures represent the real sequence (test data), the fake sequences generated by the LC model and the LC+LK model. On the same map, we indicate the number of earthquakes at each location using a heatmap (in red). Notably, the LC+LK heatmap captures the hot spot at $(32.5°N, 115.5°W)$ (green square). In contrast, the LC heatmap shows an additional hot spot at $(36°N, 118°W)$ (green circle) which is not in the real heatmap. We also es-

timate the correlation between the event count distributions of the real and the generated sequences. The LC+LK model renders a 0.936 Pearson correlation, while the LC model only scores 0.515. Therefore, using second-order statistic constraint can significantly enhance the model for generating more realistic sequences.

Model reproducibility. We make our datasets and code available at https://github.com/daDiz/LSTM2-2ndStat.

6 Conclusions

In this work, we proposed an LSTM based model for the task of modeling event sequences on network vertices. We integrated structural information of the network into conventional LSTM models to achieve improved performance. Specifically, we introduced a second-order statistic loss that measures the difference between distance distributions of the generated sequence and the target sequence. Moreover, we proposed an architecture that combines two LSTMs to learn both the slowly varying base intensities and the fast varying triggering kernels. We tested our model on synthetic and real-world datasets and illustrated its superior performance in forecasting future events.

7 Acknowledgements

This research is sponsored by the National Science Foundation (NSF) under Grant Numbers SCC-1737585, IIS-1909916 and ATD-1737996.

References

Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein GAN. *arXiv e-prints* arXiv:1701.07875.

Bengio, Y.; Simard, P.; and Frasconi, P. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5(2):157–166.

Cheng, Y.; Dundar, M.; and Mohler, G. 2018. A coupled etas-i 2 gmm point process with applications to seismic fault detection. *Ann. Appl. Stat.* 12(3):1853–1870.

Cho, K.; van Merrienboer, B.; Gülçehre, Ç.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR* abs/1406.1078.

Dixon, P. M. 2014. *Ripley's K Function*.

Du, N.; Song, L.; Gomez Rodriguez, M.; and Zha, H. 2013a. Scalable influence estimation in continuous-time diffusion networks. In Burges, C. J. C.; Bottou, L.; Welling, M.; Ghahramani, Z.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 26*. 3147–3155.

Du, N.; Song, L.; Woo, H.; and Zha, H. 2013b. Uncover topic-sensitive information diffusion networks. In *AISTATS*.

Du, N.; Dai, H.; Trivedi, R.; Upadhyay, U.; Gomez-Rodriguez, M.; and Song, L. 2016. Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, 1555–1564.

Farajtabar, M.; Du, N.; Gomez-Rodriguez, M.; Valera, I.; Zha, H.; and Song, L. 2014. Shaping social activity by incentivizing users. *CoRR* abs/1408.0406.

Frank, J., and Kingman, C. 1993. *Poisson processes*.

Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, 2672–2680.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and Harnessing Adversarial Examples. *arXiv e-prints* arXiv:1412.6572.

Graves, A., and Jaitly, N. 2014. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, II–1764–II–1772.

Graves, A. 2013. Generating sequences with recurrent neural networks. *CoRR* abs/1308.0850.

Gregor, K.; Danihelka, I.; Graves, A.; and Wierstra, D. 2015. DRAW: A recurrent neural network for image generation. *CoRR* abs/1502.04623.

Grover, A., and Leskovec, J. 2016. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, 855–864.

Hawkes, A. G. 1971. Spectra of some self-exciting and mutually exciting point processes. *Biometrika* 58(1):83–90.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation* 9:1735–1780.

Lewis, E. A., and Mohler, G. O. 2011. Research article a nonparametric em algorithm for multiscale hawkes processes.

Luo, D.; Xu, H.; Zhen, Y.; Ning, X.; Zha, H.; Yang, X.; and Zhang, W. 2015. Multi-task multi-dimensional hawkes processes for modeling event sequences. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, 3685–3691.

Ma, Y.; Guo, Z.; Ren, Z.; Zhao, Y. E.; Tang, J.; and Yin, D. 2018. Streaming graph neural networks. *CoRR* abs/1810.10627.

Marsan, D., and Lengliné, O. 2008. Extending earthquakes' reach through cascading. *Science* 319(5866):1076–1079.

Mei, H., and Eisner, J. 2016. The neural hawkes process: A neurally self-modulating multivariate point process. *CoRR* abs/1612.09328.

Mitchell, L., and Cates, M. E. 2010. Hawkes process as a model of social interactions: a view on video dynamics. *Journal of Physics A Mathematical General* 43:045101.

Mohler, G. O.; Short, M. B.; Brantingham, P. J.; Schoenberg, F. P.; and Tita, G. E. 2011. Self-exciting point process modeling of crime. *Journal of the American Statistical Association* 106(493):100–108.

Odd Aalen, O. B., and Gjessing, H. 2008. *Survival and event history analysis: a process point of view*.

Ogata, Y. 1998. Space-time point-process models for earthquake occurrences. *Annals of the Institute of Statistical Mathematics* 50(2):379–402.

Pascanu, R.; Mikolov, T.; and Bengio, Y. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, III–1310–III–1318.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, 701–710.

Plesch, A., et al. 2007. Community fault model (cfm) for southern california. *Bulletin of the Seismological Society of America* 97(6).

Ripley, B. D. 1976. The second-order analysis of stationary point processes. *Journal of Applied Probability* 13(2):255–266.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1988. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, 696–699.

SCEDC. 2013. Southern california earthquake center. Caltech.Dataset.

Short, M. B.; Mohler, G. O.; Brantingham, P. J.; and Tita, G. E. 2014. Gang rivalry dynamics via coupled point process networks. *Discrete & Continuous Dynamical Systems - B* 19(1531-3492_2014_5_1459):1459.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. *CoRR* abs/1409.3215.

van den Oord, A.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A. W.; and Kavukcuoglu, K. 2016. Wavenet: A generative model for raw audio. *CoRR* abs/1609.03499.

Xiao, S.; Farajtabar, M.; Ye, X.; Yan, J.; Song, L.; and Zha, H. 2017a. Wasserstein learning of deep generative point process models. *CoRR* abs/1705.08051.

Xiao, S.; Yan, J.; Chu, S. M.; Yang, X.; and Zha, H. 2017b. Modeling the intensity function of point process via recurrent neural networks. In *AAAI*.

Xu, H., and Zha, H. 2017. THAP: A Matlab Toolkit for Learning with Hawkes Processes. *arXiv e-prints* arXiv:1708.09252.

Xu, H.; Farajtabar, M.; and Zha, H. 2016. Learning granger causality for hawkes processes. *CoRR* abs/1602.04511.

Zhang, D.; Yin, J.; Zhu, X.; and Zhang, C. 2018. Network representation learning: A survey. *IEEE transactions on Big Data*.

Zhao, Q.; Erdogdu, M. A.; He, H. Y.; Rajaraman, A.; and Leskovec, J. 2015. SEISMIC: A self-exciting point process model for predicting tweet popularity. *CoRR* abs/1506.02594.

Zhou, K.; Zha, H.; and Song, L. 2013. Learning triggering kernels for multi-dimensional hawkes processes. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, III–1301–III–1309.

Zhu, L.; Guo, D.; Yin, J.; Steeg, G. V.; and Galstyan, A. 2016. Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE Transactions on Knowledge and Data Engineering* 28(10):2765–2777.